Probabilistic Approaches to Alert Management

Abstract

With the growing deployment of host and network intrusion detection systems, managing reports from these systems becomes critically important. Alert reports contain a variety of attributes describing, for example, the sensor's confidence in the attack, the nature of the attack, and the assets affected. Generally, not all attributes are specified on any given alert report, and management approaches should make optimal use of the attributes available. We present two components that employ probabilistic approaches to address two key problems in alert management: alert correlation and prioritization. For correlation, we extend ideas from multisensor data fusion. For shared attributes between alerts that are candidates for fusion, we consider appropriate similarity measures adjusted for situation-specific expectation of similarity. Prioritization is addressed in a Bayes model that produces an alert ranking in agreement with a human expert. By assuming that dependencies between attributes are limited to attribute groupings, the required model can be specified compactly. Our model comprehends differential weighting of attributes in an alert, incorporation of preference profiles, optimal use of observed attributes, update capability, and extensibility. We present the base model as well as a variant that can be interactively trained by a human expert. We also present results from an initial deployment of these components.

Keywords: Network security, distributed system security, alert management, alert prioritization, adaptive systems

Introduction

In response to attacks and potential attacks against enterprise networks, administrators are increasingly deploying intrusion detection systems (IDSs). These systems monitor hosts, networks, critical files, and so forth, using a variety of signature and probabilistic techniques [1, 2]. The use of such systems has given rise to another difficulty, namely, the intelligent management of a potentially large number of alerts from heterogeneous sensors. Two important aspects of alert management are alert correlation and alert priority ranking. To the degree that these aspects have been addressed in current systems, heuristic techniques have been used. We describe probabilistic approaches to these critical problems.

The intrusion detection community is actively developing standards for the content of alert messages [5]. Systems adhering to these evolving standards forward attack descriptions and supporting diagnostic data to an alert management interface (AMI), which may be remotely located and consolidating the reports of numerous sensors. In anticipation of these standards, we are evolving systems for alert correlation and ranking. The present development expands on the correlation approach in [3] and provides results of a deployment, and introduces a component that prioritizes alerts. The components rely on statistical similarity measures and probabilistic inference in the form of Bayes networks.

Bayes approaches [4], and probabilistic formalisms in general, represent a minority of inference methodologies employed to date by intrusion detection systems as well as evolving systems for correlating and prioritizing alerts from such systems. Theoretically, a probabilistic system needs to specify the entire joint probability distribution of observable attributes and corresponding priority ranking. This is extremely difficult because of the curse of dimensionality. Instead, the Bayes approach is to assume that dependencies between attributes are local, so a much more compact representation of the system's knowledge base (local conditional probability relations) is possible. The compactness of knowledge representation and the adaptive potential make this approach attractive relative to signature systems.

The remainder of this paper is organized as follows. We first expand on our earlier work in probabilistic alert correlation [3]. We then define the priority ranking mode, including a description of the adaptive training approach, wherein the desired result is to duplicate an expert's priority ranking of a given alert. We then present preliminary results from the alerts generated by various EMERALD monitors [1, 2].

Sensor Correlation and Alert Fusion

In our view, sensor correlation consists of three key functions, presented in order of increasing difficulty. At the first, we must aggregate large numbers of low-level events (TCP connections, audit records, and so forth) into a manageable number of alert reports. This is achieved by the EMERALD alert thread mechanism. At the next level, it may be desirable for some sensor to be aware of the state of other sensors, and adjust its own state accordingly. Finally, we would like to fuse alerts from heterogeneous sensors, as well as auxiliary information, to provide an operator with intelligently derived meta

alerts. Figure 1 illustrates this hierarchy; the event counts reflect actual counts from a recent 45-day run of some components on our own TCP gateway.

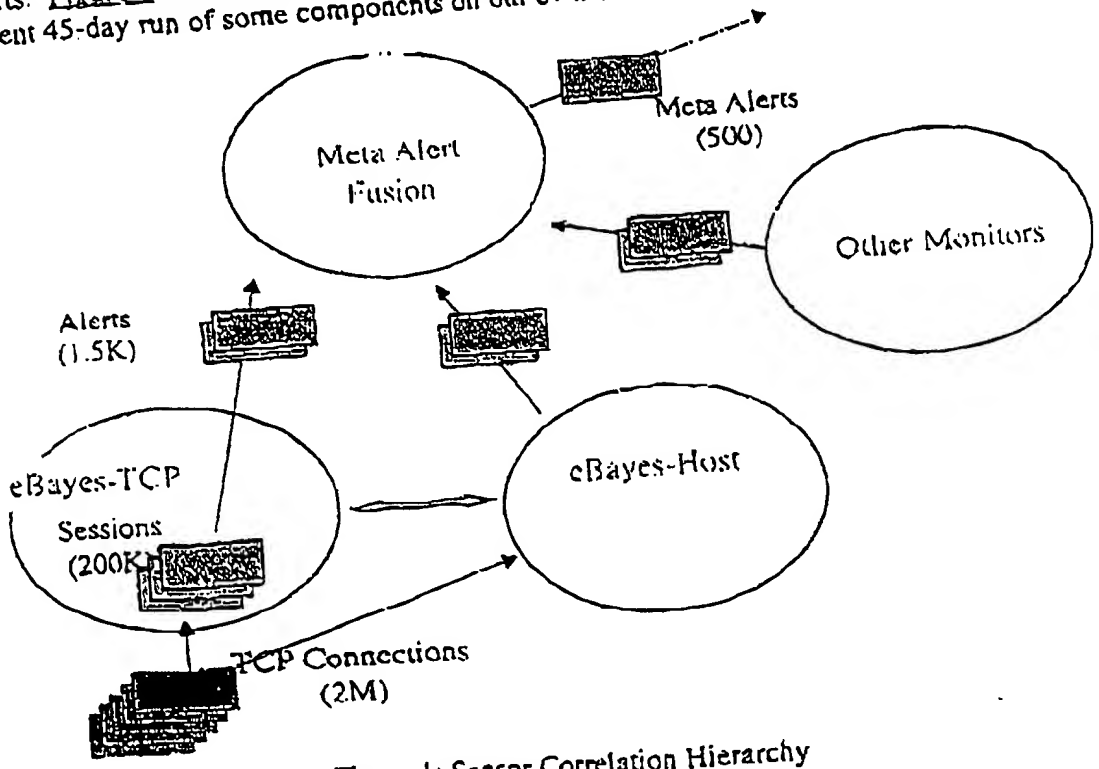

Figure 1: Sensor Correlation Hierarchy

Alert Threads

EMERALD sensors all have the concept of an alert thread, which is compatible with evolving standards for alert content [5]. Sensors may update reports on an existing attack based on transitions to more serious states, significant increases in the attack confidence level, or changes in what the sensor believes the attack to be, among other factors. For extended attacks, these updated reports may flood the analyst's console. By assigning all these reports the same thread ID, a correlation or reporting engine that comprehends threads presents the analyst a single console report, but with the capability to easily drill down if desired

Comprehending Multiple Sensors

The system we informally refer to as "eBayes" is actually two sensors: eBayes-TCP, which monitors TCP sessions, and eBayes-Host, which adapts to and monitors valid hosts and services within a protected network. These are coupled in that the former queries the state of the latter, and adjusts accordingly. This adjustment is mathematically straightforward in Bayes formalisms, which start with a prior model of their world and update this model based on the state of numerous features (directly observed or derived) linked to a set of unobservable hypotheses by conditional probability relationships.

The Bayes-Availability monitor adapts to valid hosts and services [redacted]ded on a protected subnet, and makes available to the session monitor a confidence that a given service requested is valid or not. In the latter case, the session monitor increases its prior suspicion in the case of failed requests. With this coupling, we effectively detect the stealthy portsweeps in the Lincoln Laboratory evaluation data [6]. While the small number of ports in these stealthy attacks is indicative of a specific probe for which a signature can be defined, we taken an approach that we feel is more general and powerful against new probes. The result is that we can detect a variety of other portsweeps (such as nmap and strobe) with no modification of our existing models. Thus, knowledge of the state of one sensor achieves a practical and significant improvement in the sensitivity of another.

The availability monitor also informs our session monitor when a valid service is in a degraded state (again, this is a probabilistic call). Based on this, the monitor alters its prior expectation of the world, adjusting the prior expectation of anomalous values for particular features. In a simulated attack environment, we have a large number of normal clients accessing a network that is under attack. When the attacker achieves a successful denial of service (DOS), these clients suddenly appear anomalous. With our sensor coupling, we detect the DOS and do not raise alerts for the (now failed) legitimate traffic. Essentially, the internal models dynamically come to tolerate values for some features that would normally be indicative of certain attacks. There remains the potential to generate alerts from these sessions, but the evidence must now come from other features. Without this capability, a successful DOS attack might cause hundreds of alerts for the otherwise normal traffic (what we term "collateral damage"). The large number of alerts would overwhelm the analyst, and potentially bury the one valid alert due to the attacker. The approach taken generates one alert for the attack session and one alert that the service is in a degraded state.

We therefore see that our approach, where one sensor comprehends the state of another and adjusts accordingly, both raises sensitivity (evidenced by the ability to detect stealthy portsweeps) and suppresses false (or spurious) alarms. The use of a Bayes formalism makes this coupling mathematically convenient, but is not a requirement.

Meta Alerts and the Alert Template

We are in the process of defining an enhanced alert template that enriches standards such as IDIP with respect to content while avoiding issues of language specifics. Our template includes the concept of alert thread, which is present in our monitors and alert management components. We include an "anomaly" field in addition to the "confidence" field used in IDIP. We envision these fields being used to condition the reports of some sensors. We also include arrays to describe in greater detail the target(s) of an attack (for example, the specific ports scanned). The API includes functions to allow a downstream component to know which features were set in a particular alert. Finally, we include fields describing the sensor type and placement.

This template is presently used by a diversity of sensors employing both signature and probabilistic techniques. Our early experiments indicate that diverse sensors will be able

to fill this template w[redacted]ontent that will be more useful to corr[redacted]on engines than is currently available.

Our probabilistic alert fusion approach considers feature commonality, feature similarity, and situation-specific expectation of similarity. We maintain a list of "meta alerts" that are possibly composed of several alerts, potentially from heterogeneous sensors. For two alerts (typically a new alert and a meta alert), we begin by identifying features they have in common. Such features include the source of the attack, the target (hosts and ports), the type of the attack, time information, and so forth. With each feature, we have a similarity function that returns a number between 0 and 1, with 1 corresponding to a perfect match. Similarity considers such issues as

- How well do two lists overlap (for example, list of targeted ports)?
- Is one observed value contained in the other (for example, is the target port of a DOS attack one of the ports that was the target of a recent probe)?
- If two source addresses are different, are they likely to be from the same subnet?

Not all sensors produce all possible identifying features. For example, a host sensor provides process ID, while a network sensor does not. Features not common to both alerts are not considered for the overall similarity match.

An important innovation we introduce is expectation of similarity. This is also between 0 and 1, and expresses our prior expectations that the feature should match if the two alerts are related, considering the specifics of each. For example, two probes from the same target might scan the same set of ports on different parts of our subnet (so expectation of matching target IP address is low). Also, some attacks such as SYN FLOOD spoof the source address, so we would allow a match with an earlier probe of the same target even if the source does not match (expectation of match for source IP is low).

We then compose overall alert similarity from feature similarity values normalized by expectation of similarity. The approach is valid regardless of the number of features that overlap, and is thus well suited for use with multiple heterogeneous sensors. The new alert is fused with the meta alert that is most similar, if the similarity is good enough. If no existing meta alerts are a sufficiently good match, the new alert begins a new meta alert thread. Fusion consists of combining features so that the new meta alert is a superset of the previous meta alert and the new observation. Lists are merged, comprehending "hit counts" to each list element. The merge functionality includes "trim functions" to prevent arbitrary list growth. Mathematical details of our similarity functions and expectation of similarity are given in the technical appendix, while the following subsections give more detail on feature fusion and similarity expectation.

The meta alert itself supports the threading concept, so we can visualize composing meta alerts from meta alerts.

Feature fusion

When the system decides to fuse two alerts, based on aggregate similarity across common features, the fused feature set is a superset of the features of the two alerts. Feature values in fused alerts are typically lists, so alert fusion involves list merging. For

example, suppose a probe of certain ports on some range of the protected network matches in terms of ports with an existing probe that originated from the same attacker subnet, but the target hosts in the prior alert were to a different range of our network. The attacker address list has the new attacker address appended, and the lists of target hosts are merged. The port list matches and is thus unchanged.

Rather than the common practice of listing all feature values ever seen, we maintain as well an associated hit count. For all list features we have a corresponding trim function that trims list elements with extremely low hit counts.

Two important synthetic features are the sensor and thread identifiers of all the component alerts, so that the operator is always able to examine in detail the alerts that contribute to the meta alert report. There is potential for consolidation in a useful sense, but with no loss of information since the component alerts are always available for examination.

Situation-Specific Similarity Expectation

We now give some examples of how expectation of similarity depends on the situation, that is, the features in the meta alert and the new alert.

If an alert from a sensor has a thread identifier that matches the list of sensor/thread identifiers for some meta alert, the alert is considered a match and fusion is done immediately. In other words, the individual sensor's determination that an alert is an update of or otherwise related to one of its own alerts overrides other considerations of alert similarity.

If the meta alert has received reports from host sensors on different hosts, we do not expect the target host feature to match. If at least one report from a network sensor has contributed to the meta alert and a host sensor alert is received, the expectation of similarity is that the target address of the latter is contained in the target list of the former.

To consider whether an exploit can be plausibly considered the next stage of an attack for which a probe was observed, we expect the target of the exploit (the features host and port) to be contained in the target host and port list of the meta alert.

We declare the attack code similar if we obtain an exact match, but we defer to the attack class in general. For example, in our demonstration environment, we run a variant of mscan that probes certain sensitive ports, that is, it is of the attack class "portsweep". Our host sensors have a specific signature for this attack and call it "mscan". The Bayes sensor trades specificity for generalization capability and has no "mscan" model, but successfully detects this attack as a "portsweep". Given acceptable matches in the target host and port list, these would be considered similar.

If we have observed a probe, we consider a subsequent exploit potentially similar if the target host and service (that is, port) of the exploit are contained in the target host and port lists of the probe. Of course, before fusing the alert, we require similarity with respect to other features as well.

Deciding whether the attacker is similar is somewhat more involved. In the case of an exact match of address, similarity is perfect. We assign high similarity if the subnet

appears to match. In this way, a meta alert may potentially consis[redacted] list of attacker addresses. At this point, we consider similarity based on containment. In addition, if an attacker compromises a host within our network, that host is added to the list of attacker hosts for the meta alert in question. Finally, for attack classes where the attacker's address is likely to be spoofed (for example, the Neptune attack), similarity expectation with respect to attacker address is assigned a low value.

In this fashion, it is possible to recognize a staged attack composed of, for example, a probe followed by an exploit to gain access to an internal machine, and then using that machine to launch an attack against a more critical asset.

The Priority Ranking Model

Another important aspect of alert management is the need to rank alerts, so that the administrator can concentrate on the most important alerts. We present a Bayes approach to assign a priority ranking to alerts that provide data for several key attributes. Our system has the following features:

- Ability to weight the priority ranking along several attribute groupings, such as attack type or criticality of assets affected
- Compact representation of the influence of the value of an attribute on the priority assigned.
- Incorporation of the administrator's preference profile as to the relative importance of observed values (such as attack type)
- Ranking influenced only by the attributes specified on a given alert; in general, an alert may not report all possible attributes
- Ability to update the prioritization based on observation of a new attribute
- Extensibility of the model to comprehend attributes that may be defined in the future, with minimal perturbation to the rest of the model

Computationally, our approach is to design a Bayes classifier whose output is a priority value and whose observable evidence consists of the attribute values. The influence of an attribute on the output is expressed in terms of conditional probability relations.

The system presented here addresses two problems that arise in IDS alert prioritization. The first is that, while a domain expert can assign a priority to an alert that would generally agree with that assigned by other domain experts, the volume of alerts in a realistic environment with multiple IDS makes it impossible to examine each alert in detail. Second, the depth of intrusion detection domain expertise does not at present exist in most enterprise networks. Our goal is to be able to process a large number of alerts and produce a priority ranking with which a domain expert would agree. The analogy to a "knowledge base" in a Bayes system is the set of conditional probability relations. We have taken the approach of attempting to define these according to our own domain expertise. We recognize that representation of attribute relations as conditional probabilities represents an abstraction from the expert's reasoning process. Therefore, we use this initial representation as a starting point, and provide an adaptive capability

that enables the expert to modify the system's "call". This requires domain knowledge on the part of the expert, but no special knowledge of Bayes systems.

Our model consists of a root node representing the output priority ranking (presently, "low", "medium" and "high"). From the root there are a number of main branches representing attribute groupings. These are linked to the root node by what are effectively "pass through" nodes, whose function is to weight the subtree corresponding to the principal branch. The reader will recall that this differential weighting was noted as one of the principal desirable features of our system. In the present implementation, we have two main branches connecting to nodes representing the influence of the attack attributes on the priority, while the other branch connects to a node representing the influence of asset criticality. Under the first node (node A in Figure 2) are leaf nodes expressing the relationship of attack-specific attributes to the priority. Under the second node (node B in the figure) are leaf nodes representing criticality of asset classes that may be touched by an alert. This representation of attribute groupings as major branches from the root allows for subjective weighting as to the relative importance of the attributes in question. The relative weighting of the major branches is achieved by the "pass through" action of nodes A and B, as discussed in more detail below

We have proposed this model as a demonstration of concept, namely, that attributes can be grouped and that we wish to potentially express the fact that different groups should have different impacts on the output result. The proposed two attribute groups, described in the next subsections, are only one possible division. We are not limited to two attribute groups, nor are we precluded from a more complex branching structure within the attribute groups. Our proposed model seems to provide a good tradeoff between simplicity and efficacy.

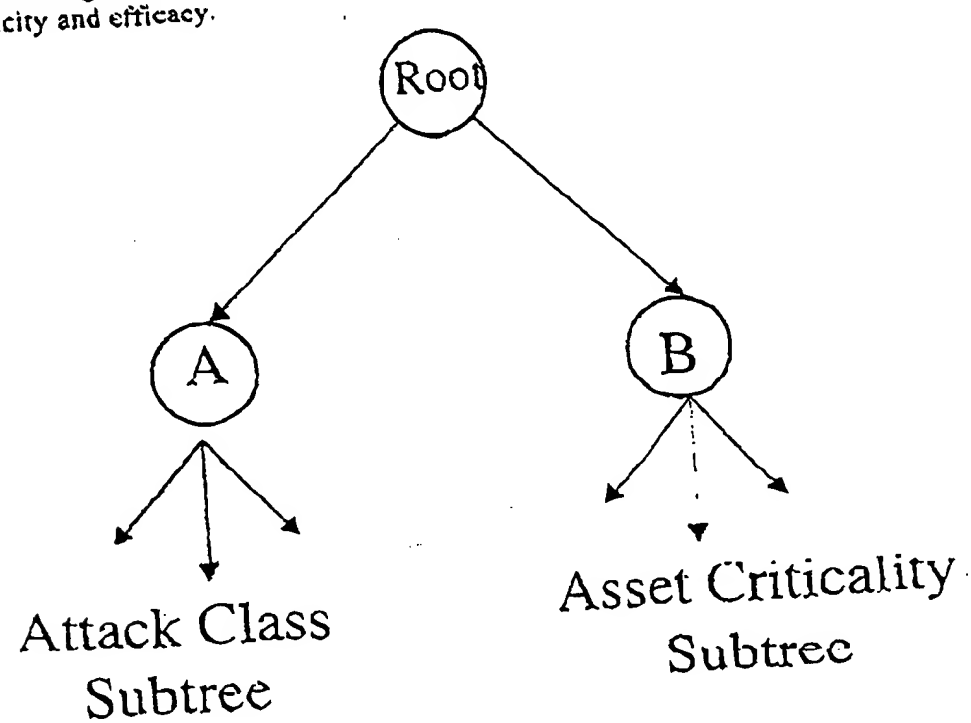

Figure 2: Bayes Model for Alert Prioritization

Attack Class Attributes

At present, attack class attributes are expressed as a preference profile over the attack classes. This is a configurable measure of concern preassigned to each attack class. We configure our system so that probes are of low concern, while attacks such as unauthorized access are of greater concern. An alert (raw or fused) provides a probability mass over the attack classes; systems capable of a single call assign all their mass to one attack class. This is potentially moderated based both on the sensor's confidence in its call as well as the fusion and prioritization engine's confidence in the sensor. Based on this probability mass, we perform the equivalent of an expected value calculation and return a priority score based on attributes of the attack class. The system has a node that relates this partial score to the overall priority assigned at the root node.

Asset Criticality Attributes

Figure 3 represents the subtree that relates asset criticality to alert priority. Our initial model identifies five attributes that are potentially critical: user, protocol, service, file (to include directory), and host/subnet. A given alert may not provide values for all these attributes. For example, the TCP session monitor in [2] does not examine user or file. Therefore, for any of these attributes, the alert processor passes to our model one of three values: the attribute was not observed, the attribute was observed and not considered critical, and the attribute was observed and critical. Criticality of an asset is based on a configuration file that reflects security policy. Our model supports dynamic change to the security policy. The elements of the respective CPTs reflect $P(criticality = c | priority = p)$. Each of these matrices represents two values of criticality by three values of priority. Therefore, the local knowledge base consists of a set of CPTs linking the attribute to the appropriate node on its main branch. If the attribute is not observed in a given alert, the state of the corresponding node is not changed, and thus this attribute does not influence the result one way or the other. If this attribute is observed in a subsequent update for the same alert, our system adjusts the previous prioritization for the new information. Compact representation, use of only the observed attributes, and update of an existing result to comprehend new information are all key desirable features of a system of this type, and they are rigorously addressed in our formalism.

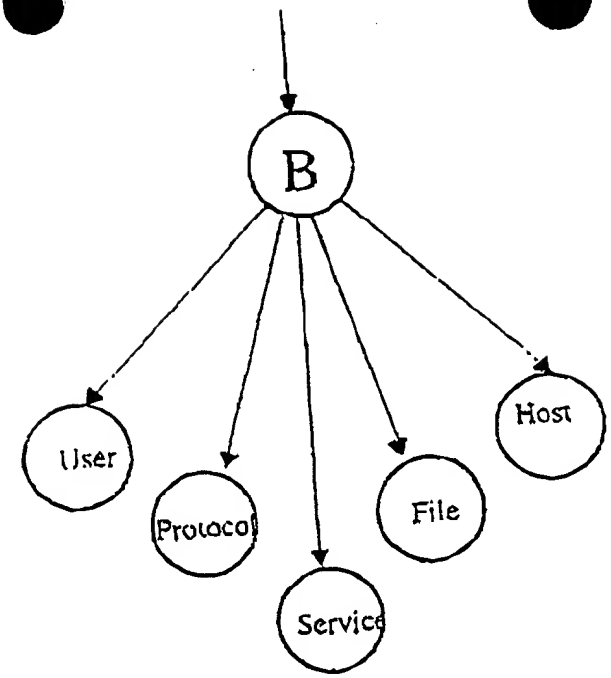

Figure 3: Asset Criticality Subtree

Pass-Through Nodes

Nodes A and B are the roots of distinct subtrees reflecting the influence of different groups of attributes on the desired result. We can consider these nodes as serving a "pass through" function, propagating the subtree result to the root. If the CPTs relating these major branch nodes to the root are identity matrices, the evaluation from the leaves under the branch is passed through without alteration. Moving mass off the diagonal effectively perturbs and downweights the corresponding subtree result, expressing, for example, different subjective assignments of importance or confidence in a particular attribute group. Initially, these CPTs are near-identity matrices, with some off diagonal mass.

Adaptive Training

The Bayes inference engine used here has an adaptive capability described in [2]. Briefly stated, the system behaves as if the CPTs are based on effective counts. If a hypothesis (priority assignment) "wins" (posterior belief above a settable learning threshold), entries in the CPT are adjusted slightly in the direction of the observation (which are really the likelihood messages at the leaf nodes). The effective count for the winning hypothesis is aged (multiplied by a decay factor) and incremented by one for the current observation. The effective counts for other hypotheses are aged. Therefore, frequently observed hypotheses approach a saturation count, and as the adjustment depends on the effective count, new observations perturb the current CPT only slightly. Conversely, a very rarely observed hypothesis adapts more quickly to new observations, as its effective count decays to a lower value and thus assigns less weight to past values. This can be thought of as hypothesis-specific annealing.

To utilize this in the present system, we have an interactive facility that randomly generates attribute values and prompts the operator for the priority ranking to be assigned to the alert in question. This becomes the state value for an ancillary "hard call" node, whose value forces the call for the net as a whole. The learning facility is then invoked, and the CPT values adapt accordingly. We believe that with reasonable initial values for the CPT, based on our expert judgment in this area, this iterative adaptation can be achieved with far fewer simulated alerts than would be required to, for example, train a neural net from scratch.

Results

Live Data

The following result is from a probe attack detected against our internal network that apparently looked for various ports on different subnets over several days. Each individual component of the attack is detected by the Bayes TCP sensor, which does not know beforehand that this is a sensitive set of ports. The attack does not cause a denial of service.

```
portsweep   1.000 2000-08-10 06:50:28 from          193.230.37.2 ports
1268 to 32434 dt=   0.321
  count    164 max age count   0.16 code    3 svc    1 max-err  3.83 -
opn    0 -oip    0 -oport    0
  30 dest IPs: aaa.bbb.1.1 aaa.bbb.3.1 aaa.bbb.4.1 aaa.bbb.5.1
aaa.bbb.6.1 aaa.bbb.7.1 aaa.bbb.8.1 aaa.bbb.9.1 aaa.bbb.10.1 aaa\
.bbb.11.1 aaa.bbb.12.1 aaa.bbb.13.1 aaa.bbb.14.1 aaa.bbb.15.1
aaa.bbb.16.1 aaa.bbb.17.1 aaa.bbb.18.1 aaa.bbb.19.1 aaa.bbb.20.1 \
aaa.bbb.21.1 aaa.bbb.22.1 aaa.bbb.23.1 aaa.bbb.24.1 aaa.bbb.25.1
aaa.bbb.26.1 aaa.bbb.27.1 aaa.bbb.28.1 aaa.bbb.29.1 aaa.bbb.30.1
aaa.bbb.31.1
  6 dest ports: 635(30) 110(29) 143(29) 53(27) 21(27) 109(22)
  BEL          0.000    0.000    0.000    0.000    0.000    0.000
0.000    0.000    0.000    0.994    0.004    0.002    0.000
  PCODE        0.000    0.000    0.000    1.000    0.000    0.000
0.000
  SVC DIST     0.417    0.000    0.194    0.000    0.389    0.000

Non-sys alloc ports 6  port_anom 0.990528 code_anom 0.987018
Invalid hosts 30  Invalid ports 6  Neval 24 eval_count 0

LL-LIST 2000-08-10 06:50:28  aaa.bbb.1.1 to aaa.bbb.31.1  portsweep
1.000
```

Similar attacks with different target IP addresses appeared over several days. The meta alert fusion utility produced the following meta alert:

```
Meta_alert thread 22
Source IPs 193.230.37.2

Target IPs aaa.bbb.1.1 aaa.bbb.3.1 aaa.bbb.4.1 aaa.bbb.5.1 aaa.bbb.6.1
aaa.bbb.7.1 aaa.bbb.8.1 aaa.bbb.9.1 aaa.bbb.10.1 aaa.bbb.11.1
aaa.bbb.12.1 aaa.bbb.13.1 aaa.bbb.14.1 aaa.bbb.15.1 aaa.bbb.16.1
aaa.bbb.17.1 aaa.bbb.18.1 aaa.bbb.19.1 aaa.bbb.20.1 aaa.\
```

```
bbb.21.1 aaa.bbb.22.1 aaa.bbb.23.1 aaa.bbb.24.1 aaa.[redacted]25.1
aaa.bbb.26.1 aaa.bbb.27.1 aaa.bbb.28.1 aaa.bbb.29.1 aaa.bbb.30.1
aaa.bbb.31.1 aaa.bbb.1.2 aaa.bbb.3.2 aaa.bbb.4.2 aaa.bbb.5.2
aaa.bbb.6.2 aaa.bbb.7.2 aaa.bbb.8.2 aaa.bbb.9.2 aaa.bbb.10.2 aaa.bb\
b.11.2 aaa.bbb.12.2 aaa.bbb.13.2 aaa.bbb.14.2 aaa.bbb.15.2 aaa.bbb.16.2
aaa.bbb.17.2 aaa.bbb.18.2 aaa.bbb.19.2 aaa.bbb.20.2 aaa.bbb.21.2

From  2000-08-10 06:50:28 to  2000-08-14 01:47:18

Ports
Sum Hits 708.999  dot product 0.166775
Index 635    Prob 0.170455
Index 110    Prob 0.168534
Index 143    Prob 0.167341
Index 53    Prob 0.165906
Index 21    Prob 0.169958
Index 109    Prob 0.157806

Threads :4860 5564 6314 6980 7650 8223 8767 9287 9778 10350 10964 11577
12188 12634 13033 13427 13802 14166 14525 14887 15298 15677 16015 16357
16689 17040 17321 17584 17907 18238 18558 18861 19185 19607 19980

Attack steps: portsweep
```

This sequence of attacks was confirmed as a virtually certain attack by our system administration staff. The list of thread identifiers permits the administrator to examine any of the stages in the attack. In this case, each attack stage is considered a portsweep; if the stages consisted of different attack classes, these would be listed under "Attack steps" above. This attack was a probe (low priority) but accessed at least one critical host, so the overall assigned priority is 0.45 (on a scale of 0 to 1).

Simulation Environment

We describe the operation of the above components in a simulated attack environment. The environment simulates an electronic commerce site, and provides Web and mail services over the Internet. The protected network is behind a firewall, and is instrumented with several host, network, and protocol monitors. Two machines are visible through the firewall, and two auxiliary machines are used for network monitors and alert management. The firewall blocks all services other than Web and mail. We have simulated traffic that appears to come from a number of sources, and an attacker who executes certain well-known attacks.

The attack begins with an mscan probe to the two machines visible through the firewall. Signature-based sensors on the two machines detect this and set the attack code for mscan in their alert report. The Bayes network sensor detects the attack as of the class "portsweep". The target port lists for these alerts match, as does the attacker address. Similarity with respect to the target address depends on the order in which the alerts arrive (the order at which the alerts arrive is not deterministic as the sensors are distributed). Until the network sensor's alert is received, we do not expect target addresses to match for distinct sensors, since a host sensor can typically report only its own address as a target. We expect the network sensor's target list to contain the list of targets seen so far, and we expect the target address of subsequent host alerts to be

contained in the meta alert's target host list. Regardless of arrival order, these alerts are fused. We have set a preference profile value of LOW (0.1) for probe attacks, but since one critical host and one critical service were probed, the overall alert is prioritized by the Bayes prioritization component at 0.64 on a scale from 0 to 1.

The attacker next uses a CGI exploit to obtain the password file from the Web server. This is detected by the host sensor. The fusion engine considers this a new step in the existing attack because of a match in attacker address as well as the fact that the target host and port are contained in the target and port list of the existing probe. Criticality rises to over 0.95 because of the attack escalation (probe to unauthorized access) and asset criticality (critical host, service, and now file).

The next stage of the attack is a buffer overflow to gain access on the host providing mail service. This is detected by the host sensor. Once again, the attack target is already included in the list of the meta alert, so this alert is fused as well. It is important to note that the address of this compromised host is added to the list of attacker source hosts (that is, it is both a target and now a potential source for an attack).

Although telnet through the firewall is blocked, the attacker may telnet from the compromised internal host to the critical host, and he now does so. On that host, he uses another overflow attack to obtain root access. He is now free to change Web content. All these actions are detected, and fused into one meta alert expressing all stages of the attack. Criticality at this point is extremely high (essentially 1.0).

It is worth pointing out that all the sensors in question provide response directives which, if followed, would stop the above attack in progress.

Summary

As intrusion detection systems are more widely deployed, the problem of alert management assumes paramount importance. This is because a qualified domain expert is not able to examine all alerts produced on realistic enterprise networks by present IDSs, and at any rate such domain expertise is still not widespread. We have proposed a system that addresses two aspects of alert management, namely, alert correlation and prioritization. The probabilistic approach of these systems gives an alternative and complement to the heuristic techniques more commonly used in intrusion detection.

We have adapted and extended notions from the field of multisensor data fusion for alert correlation. The extensions are principally in the area of generalizing feature similarity functions to comprehend observables in the intrusion detection domain. The approach has the ability to fuse alerts for which the match is good but not perfect. A list of identifiers is maintained with each meta alert that permits linkage to the individual component alerts.

Our approach to prioritization is to construct a Bayes network whose observables are the attributes potentially reported for an alert and whose output is an alert priority. The knowledge base is compactly represented as local conditional probability relations between members of attribute groups and between these attribute groups and the output priority. The system has an adaptive capability wherein a domain expert can modify the system's "call" for randomly generated alert exemplars. This training facility can be

invoked for the system as a whole or for portions of the system corresponding to major attribute groupings, represented as Bayes subtrees.

Acknowledgments

This research was sponsored by DARPA under contract number F30602-99-C-1049. The views herein are those of the author(s) and do not necessarily reflect the views of the supporting agency.

References

1. Porras, P. and Neumann, P. "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances", National Information Security Conference, 1997. http://www.sdl.sri.com/emerald/emerald-niss97.html

2. Valdes, A. and Skinner, S. "Adaptive, Model-based Monitoring for Cyber Attack Detection", Recent Advances in Intrusion Detection (RAID 2000), Toulouse, France, October 2000. http://www.raid-symposium.org/raid2000/program.html

3. Valdes, A. and Skinner, S. "Blue Sensors, Sensor Correlation, and Alert Fusion", Recent Advances in Intrusion Detection (RAID 2000), Toulouse, France, October 2000. http://www.raid-symposium.org/raid2000/program.html

4. Pearl, J. "Probabilistic Reasoning in Intelligent Systems", Morgan-Kaufmann (1988).

5. Erlinger, M. and Stanniford, S. "Intrusion Detection Interchange Format", http://www.ietf.org/html.charters/idwg-charter.html

6. Lippmann, Richard P, et al. "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-Line Intrusion Detection Evaluation," Proceedings of DARPA Information Survivability Conference and Exposition, DISCEX'00, Jan 25-27, Hilton Head, SC, 2000. http://www.ll.mit.edu/IST/ideval/index.html

Technical Appendix: Mathematical Description of Similarity and Similarity Expectation

Similarity

In the following discussion, X is defined to be the value of a feature in a new alert, and Y is the value of that feature in an existing alert class to which it is being compared. For features that have a single value, the features of a new alert and an existing alert class are defined to be similar if their feature values are equal. That is,

$$SIM(X,Y) = \begin{cases} 1.0, X = Y \\ 0, otherwise \end{cases}$$

The more general case is one in which a feature of an alert includes a list of observed values, and a "hit count" of the number of times each value has been observed. For reasons of normalization, the hit count may be converted to a probability. In this sense, X and Y are lists of feature and probability values, possibly of different lengths. A probability vector describes a pattern over the observed categories, where

$p_X(C)$ = probability of category C in list X

$p_Y(C)$ = probability of category C in list Y

P_X = probability vector over categories observed for X

P_Y = probability vector over categories observed for Y

Note that these probability vectors are more general than the typical binary representation frequently employed in feature matching. The notation $C \in X$ to denote that category C occurs in list X. Then we can use a variety of similarity functions based on normalized dot products (or equivalently, cosine separation of feature vectors) such as

$$Sim(X,Y) = \frac{\left[\sum_{C \in X \text{ AND } C \in Y} P_X(C) \times P_Y(C)\right]^2}{(P_X \cdot P_X)(P_Y \cdot P_Y)}$$

If the two lists are the same length, then this measure gives the square of the cosine between the two. This has an intuitive geometric property that is not shared by, say, the dot product as is commonly used in the pattern matching literature. If the patterns (1, 0), (0, 1), and (0.5, 0.5) are understood to be over the same two categories, then

$Sim(\{1,0\},\{0,1\}) = 0$

$Sim(\{0.5,0.5\},\{0,1\}) = Sim(\{0.5,0.5\},\{1,0\}) = 0.5$

In other words, the first two patterns are orthogonal, and the third is halfway in between.

Similarity measures of this type are typically found in multisensor data fusion applications. For our purposes, we must extend this notion of similarity for some features. For example, there are special cases of similarity where we want to consider an observed feature "similar" if it is included in the list of the candidate alert class to which

it is being compared. [redacted]robe attack may search for a vulnerable [redacted]vice on some host, and then an attack to a specific host exploits the vulnerability. [redacted] case, we consider the target "similar" if the target of the exploit is included in the target list of the probe. In addition, IP addresses have special similarity functions to express an intermediate value for addresses appearing to originate from the same subnet. Finally, in the case where a target host is compromised, its address is added to the attacker's address list, and is a candidate for similarity evaluation as an attack originator.

Expectation of Similarity

As was discussed earlier, the nature of an alert may change an expectation of which features of a new alert and an existing alert class should be similar. For example, a syn flood is a type of attack in which the source IP address is typically forged. In this case, similarity in the source IP address would not be considered when assessing the overall similarity between a new alert triggered by a syn flood attack and an existing alert class.

Expectation of similarity is a feature-specific number between 0 and 1, with 1 indicating a strong expectation of a match, and 0 indicating that two otherwise similar alerts are probably not similar with respect to a specific feature. An alert state is represented as a probability vector over candidate states. For each candidate alert state, each feature has a vector of the same length whose elements are the similarity expectation values given the state. This table may be initially populated with MEDIUM (about 0.6) values, except where it is known otherwise. For example, in an access from a compromised host or in the case of a DOS, there is a LOW expectation that the source IP will match that of the attacker.

Based on the alert state, the similarity expectation may be dynamically generated as the weighted sum of the elements of an expectation table, with the weights from the (evolving) alert state distribution. This is itself an expectation in the statistical sense, conditioned on the belief over the present alert state. This is general enough to cover the situation of an unambiguous call from a signature engine, in which case the distribution over states is 1.0 at the state corresponding to the call and 0 elsewhere. Algebraically, for a feature J

$$E_J = \sum_{i \in \{alert\_states\}} BEL(i) E_J(i)$$

E_J = expected similarity for feature J, given present state

$BEL(i)$ =belief that the attack state is presently i

$E_J(i)$ = element i of the lookup table of expected similarity for measure J given state i

Both the new alert and the alert class to which it is being compared each compute the similarity expectation for each feature. Feature similarity and similarity expectation are then combined to form a single value of alert similarity. This is done by combining feature similarities and normalizing by similarity expectation, over the set of common features:

$$SIM(X,Y) = \frac{\sum_J E_J^X E_J^Y \mathrm{sim}(X_J, Y_J)}{\sum_J E_J^X E_J^Y}$$

X =new alert

Y =existing alert

J indexes features

$E_J^{X(Y)}$ = similarity expectation for feature J, alert X(Y)

A similar definition can be formed by taking products rather than sums above, and then taking the geometric mean. This has some advantages, but can be overly influenced by a single extremely small value.

Global changes:

' '	to	‘ ’
" "	to	“ ”
multi-sensor	to	multisensor
port scan	to	portsweep
net	to	network
subnet	to	subnetwork
internet	to	Internet
web	to	Web

Be consistent throughout: portsweep, port sweep, port scan, stealthy portsweep, stealth port sweep,

Be consistent: thread identifier or thread ID

At first use, establish IDS as an abbreviation for intrusion detection system and use thereafter, or spell out in full throughout the paper.

If ID is used for "identifier", do not use for "intrusion detection" (unless as part of IDS).

Be sure that TCP and CPT are not confused, and define CPT at first use.

Check "set of CPT" --should it be "set of CPTs"? Note that set needs a singular verb: "set is" Note that CPT needs a singular verb: "the CPT is", but "these CPTs are"

Run spell check.